

LESSON 1: INTRODUCING 8051SBC MICROPROCESSOR LEARNING BOARD

Objectives:

1. Enter machine code, using single step to execute 8051 instructions.
2. Examine registers.
3. Test run simple program that blinks onboard LED.

The 8051SBC uses standard serial port, RS232 for terminal interfacing. The format of RS232 is 9600 bit per second, 8-data bit, no parity, and one stop bit, no flow control. We can use PC running communication program to emulate terminal functions. When we press keyboard, the ASCII code will send to the 8051SBC through RS232 port. When the 8051SBC sends the ASCII code back to PC, it will show on terminal screen.

Let's connect the 8051SBC to PC COM port and press reset, we get,

8051SBC Microprocessor Learning Board

ADDR:8000>

After reset or power up the board, the 16-bit Program Counter register will reset to 0000H. The CPU will start fetch the code from that location. The 8051SBC stores the small program called monitor program at address 0000H to 7FFFH. The monitor program helps user to enter code, examine register and run the user program. The prompt ADDR:8000> is produced by such monitor program running.

Now type ? for commands listing.

ADDR:8000> Help

Standard Commands

- ?- This help list
- M- List programs
- R- Run program
- L- Download
- U- Upload
- N- New location
- J- Jump to memory location
- H- Hex dump external memory
- I- Hex dump internal memory
- E- Editing external ram
- C- Clear memory

User Installed Commands

- D- Disassemble
- S- Single-Step

Q- Quick home
Z- ZAP EEPROM
W- NEW COMMANDS

ADDR:8000>

As we know that the memory space 64kB is divided into two pages, each has 32kB. The first page is ranging from 0000H to 7FFFH and the second page is from 8000H to FFFFH. The first page is ROM that stores monitor program. The second page is RAM. We will use the second memory page for testing our program.

Suppose we want to test small program as shown below.

ADDR OP CODE

```
8100 B297      main: cpl p1.7
8102 00                nop
8103 80FB                sjmp main
```

The command E helps editing external RAM, type 'e' at prompt. Enter new value to the ram.

ADDR:8000> Editing external ram

Address> 8100

8100: (BE) New Value: B2

8101: (95) New Value: 97

8102: (E3) New Value: 00

8103: (21) New Value: 80

8104: (FB) New Value: FB

8105: (F8) New Value: Editing complete, this location unchanged

ADDR:8105>

The value shown in parenthesis is current data in ram. If we type new value, B2 will enter into the ram. B2 will be written over BE.

We can dump the content of memory with command 'h'. However the current location is now 8105, we must change it to 8100 with command 'n' new location.

ADDR:8105> New location

New memory location: 8100

ADDR:8100>

Then type command 'h' hex dump.
ADDR:8100> Hex dump external memory

```
8100: B2 97 00 80 FB F8 E6 51 FC A1 01 04 F6 C7 21 15 2 {xfQ|! vG!  
8110: FF 4C DA E2 DF ED FD 03 63 22 CD 01 A5 08 C6 C0 LZb_m} c"M % F@  
8120: 59 79 EF AA 13 33 F3 B3 9D 13 3B 68 A1 02 F4 85 Yyo* 3s3 ;h! t  
8130: 7B 31 6F 4D DE 01 7D F1 96 24 30 04 05 01 4B 14 {1oM^ }q $0 K  
8140: CF 80 FF A3 FB 43 F7 6E 16 40 F4 78 56 52 17 2A O #{Cwn @txVR *  
8150: D5 59 EF D5 F7 EC BE 9D 94 20 44 8C 23 80 4E 86 UYoUwl> D # N  
8160: FF 5D ED 79 FF B9 AF 23 2C C7 47 81 CE 05 9C 56 ]my 9/#,GG N V  
8170: FA D0 ED DB 8E 6D A6 EA 1C 83 DA 18 43 D4 00 10 zPm[ m&j Z CT  
8180: 75 F4 FF 4C BF F0 FE AA 21 20 95 E8 04 6A 7E 04 ut L?p~*! h j~  
8190: A1 21 72 EF FE EE BE F4 71 56 82 96 F1 02 BC 00 !!ro~n> tqV q <  
81A0: EB C1 BD B9 8F 02 77 4D 25 26 02 02 5A 98 82 80 kA=9 wM%& Z  
81B0: F7 AC BF DB FF FF FB 12 5B 00 D4 80 D2 72 65 29 w,?[ { [ T Rre)  
81C0: CB B7 6D 2D FF 4D B8 FD 0D E6 4E A6 4E B8 A7 20 K7m- M8} fN&N8'  
81D0: F7 65 E7 CE BF 79 DD 78 11 20 44 81 57 00 B6 BC wegn?y)x D W 6<  
81E0: 8B 5B 6C 28 EF AE F7 4E 55 C1 CE 01 35 30 18 5C [l(o.wNUAN 50 \  
81F0: EE FF FC FE BE 91 97 8A 1B 02 91 18 5F 6E 37 9E n |~> _n7
```

ADDR:8200>

We see that the content of locations 8100 to 8104 are now changed to our code, B2 97 00 80 FB. The Left-hand is location or address of external RAM. Each line contains 16 bytes, from 8100 to 810F. The Right-hand is the ASCII code representation for each byte.

Try with new command, 'd' disassemble. See what happen. Change location to 8100 and type command 'd'.

ADDR:8200> New location

New memory location: 8100

ADDR:8100> Disassemble

```
8100: B2 97    CPL    P1.7  
8102: 00        NOP  
8103: 80 FB    SJMP  8100  
8105: F8        MOV   R0, A  
8106: E6        MOV   A, @R0  
8107: 51 FC    ACALL 82FC  
8109: A1 01    AJMP  8501  
810B: 04        INC   A  
810C: F6        MOV   @R0, A  
810D: C7        XCH  A, @R1  
810E: 21 15    AJMP  8115
```

```

8110: FF      MOV   R7, A
8111: 4C      ORL   A, R4
8112: DA E2   DJNZ  R2, 80F6
8114: DF ED   DJNZ  R7, 8103
8116: FD      MOV   R5, A
8117: 03      RR    A
8118: 63 22 CD XRL   22, #CD
811B: 01 A5   AJMP  80A5
811D: 08      INC   R0

```

ADDR:811E>

Disassembler is a small program that converts opcode or machine code into mnemonic and operator. We see that our codes are converted into 8051 instructions.

We can test our program by executing each instruction one by one. The method is called single step running. To enable it, place the jumper that ties INT1 to GND. Find the onboard jumper labeled, “Single Step”.

Change new location to 8100 and type command ‘s’ for single stepping.

ADDR:8100> Single-Step

Jump to memory location (8100), or <ESC> to exit:

Now running in single step mode: <RET>= step, ?= Help

```

ACC B C DPTR  R0 R1 R2 R3 R4 R5 R6 R7  SP  Addr Instruction
00 00 0 8100  00:00:00:00:00:00:00:00 0A  14DD: JMP   @A+DPTR
00 00 0 8100  00:00:00:00:00:00:00:00 0A  8100: CPL   P1.7

```

Now you can see that the onboard LED will turn on. Press ENTER to execute next instruction.

```

00 00 0 8100  00:00:00:00:00:00:00:00 0A  8102: NOP
00 00 0 8100  00:00:00:00:00:00:00:00 0A  8103: SJMP  8100
00 00 0 8100  00:00:00:00:00:00:00:00 0A  8100: CPL   P1.7

```

We see that when CPL 1.7 was executed again, the LED will turn off.

We can use single step running for learning new instruction and examine the content of registers. Below show the name of registers that we can examine.

ACC B C DPTR R0 R1 R2 R3 R4 R5 R6 R7 SP

ACC is 8-bit accumulator,

B is register B,

C is 1-bit carry flag stored in PSW (Program Status Word register),

DPTR is 16-bit data pointer,
R0-R7 are 8-bit general purpose registers, and SP is 8-bit stack pointer.

EXERCISE 1: Replace NOP instruction with instruction INC A. The opcode of instruction INC A is 04. Single step the program and answer the following questions.

1. Where is the address to be placed opcode 04?
2. What is the content of accumulator after instruction INC A was executed? What is the maximum value?

Now someone may wonder why we do not use command 'j' to jump from monitor program to our program? Let's try new command 'j'. See what happen.

ADDR:811D> Jump to memory location

Jump to memory location (811D), or ESC to quit: 8100

running program:

Notice the onboard LED, does it turn on?

We see that the LED is turned on, but not blink as we have been using single step. We know that the operation of microprocessor is FETCH-EXECUTE cycle. Each instruction has different number of cycle. Most of instruction needs only one cycle, but some of them need two or three cycles. The CPU data book will show details for each instruction. Let's take a look the sample instruction, CPL P1.7.

CPL bit

Byte: 2

Cycles: 1

Encoding:

1 0 1 1 0 0 1 0

bit address

Operation: CPL

$(bit) \leftarrow \neg (bit)$

One machine cycle contains 12 clocks. For example if we use 12MHz oscillator, one machine cycle will be 1 microsecond.

EXERCISE 2: Draw the logic level appears at P1.7 when running the code with command 'j'. (NOP = 1 cycle, SJMP = 2 cycles) What is the period of logic high and logic low? (the XTAL oscillator is 12MHz)

I do remember someone in the class asked me to add the delay subroutine after bit P1.7 was complemented. This makes the blinking of LED slower rate, we can make it blink likes single step running but with command jump instead.

Here is the sample delay subroutine.

```
ADDR OPCODE
8105 7F00      delay: mov r7,#0
8107 DFFE          djnz r7,$
8109 22          ret
```

We use R7 as the loop counter. DJNZ will decrement R7 by one and check if R7 equals 0 or not. If R7 does not equal 0, it will repeat decrement again. But if it equals 0 the loop will end.

Add the subroutine into our first program.

```
ADDR OPCODE
8100 B297      main:  cpl p1.7
8102 128107    call delay
8105 80F9      jmp  main

8107 7F00      delay: mov r7,#0
8109 DFFE          djnz r7,$
810B 22          ret
```

EXERCISE 3: Enter the code into memory from address 8100 to 810B with command 'e'. Single step from address 8100. Where is the address of subroutine delay? What is the result in R7 after DJNZ was first time executed? How many times the instruction DJNZ was repeated?

Note: The opcode of instruction jmp main has only two bytes, 80F9, where 80 is instruction code, F9 is called OFFSET byte. The operation of jmp main, will force CPU to jump back to location 8100. After the CPU fetches two bytes, 80 and F9 the content of Program Counter register will be 8107. When the jmp instruction was executed, the program counter 8107 will be added with FFF9, resulting 8100. So next byte to be fetched will be B2 again.

EXERCISE 4: Show how to find where is the location that DJNZ will be repeated if the content of R7 is not 0?

EXERCISE 5: Run above program with command jump. Notice the LED, does it blink? How can we make the LED to be blinked slower frequency? Below is sample nest delay loop. Try modifying the code to make the LED blinks with full speed running!

```
ADDR OPCODE
8107 7F0A      delay: mov r7,#10
8109 7E00      delay1: mov r6,#0
810B DEFE          djnz r6,$
810D DFFA          djnz r7,delay1
810F 22          ret
```